

## RM17xx 说明书



[www.rockmong.com](http://www.rockmong.com)

上海岩獾科技有限公司  
V1.0

## 目录

RM17xx 说明书 .....	1
一、 产品特点 .....	3
二、 产品选型 .....	3
三、 主要参数 .....	3
四、 安装方式 .....	3
五、 接线方式 .....	4
六、 驱动 .....	4
七、 调试工具使用介绍 .....	4
1. 工具总览 .....	4
2. 数字 I/O .....	6
2.1 引脚模式介绍 .....	6
2.2 状态介绍 .....	6
3. 修改上电状态 .....	7
4. 修改 SN .....	7
八、 基础编程一介绍——单个 IO 操作 .....	8
1. 获取设备 .....	8
2. 读取输入状态 .....	8
3. 控制输出状态 .....	8
4. 读取输出状态 .....	8
九、 基础编程二介绍——多个 IO 同时用 bit 操作 .....	9
1. 读取输入状态 .....	9
2. 控制输出状态 .....	9
3. 读取输出状态 .....	9
十、 其他编程介绍 .....	10
1. 多个输入口同时读取 .....	10
2. 多个输出口同时写入 .....	10
3. 多个输出 IO 同时读取状态 .....	11

## 一、产品特点

- 电源：DC 7-30V；
- 输入输出：多路光耦隔离数字输入；继电器输出；
- 通讯接口：USB；
- 通信速率：一次读写只需 0.5ms（具体受电脑环境等因素限制）；
- 通信协议：无需关心底层实现，调用库函数即可，多线程安全，易移植，使用简单，高效；
- 跨平台：支持 Windows、Linux、Android、Mac OS；
- 提供各大语言例程：C/C++、C#、Python、Java、LabView 等等；
- 支持修改上电时输出状态；

## 二、产品选型

型号	输入输出
RM1702	二路输入二路输出
RM1704	四路输入四路输出
RM1708	八路输入八路输出
更多通道请联系我们	

## 三、主要参数

参数	说明
电源额定电压	DC 7-30V
运行指示灯	绿色 LED 常亮表示正常运行，熄灭表示 USB 连接断开
输入	电源 24V 时：逻辑高 18~24V，逻辑低 0~18V 电源 12V 时：逻辑高 6~12V，逻辑低 0~6V 电源 9V 时：逻辑高 6~9V，逻辑低 0~6V
输入指示	红色 LED 指示
继电器输出	5A@30VDC；5A@250VAC；10A@125VAC
输出指示	红色 LED 指示
通讯接口	USB
通讯速率	最大 2KHz
温度范围	工业级，-40℃~85℃

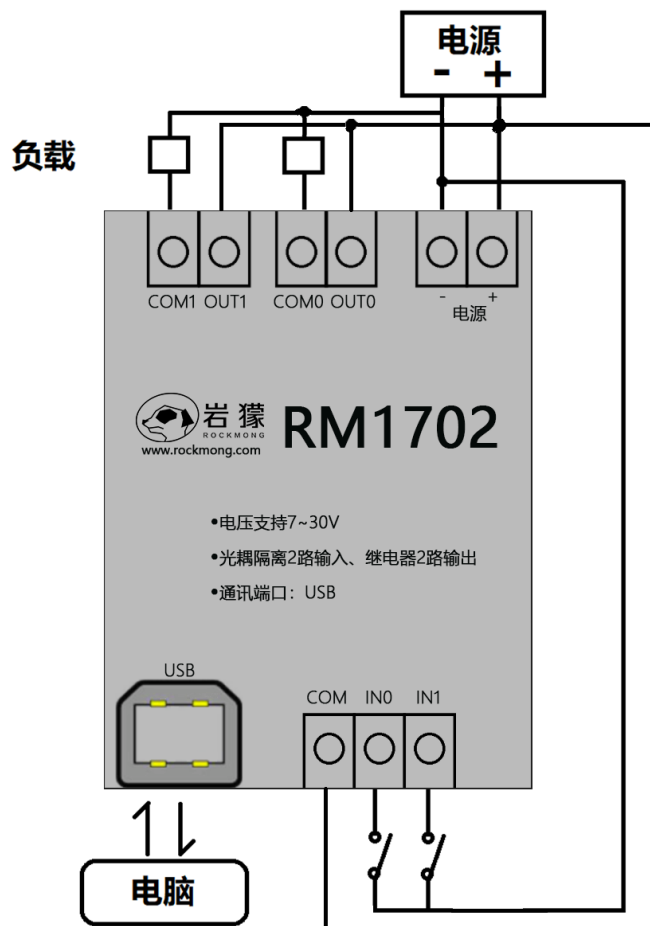
## 四、安装方式

工控盒型、导轨、螺纹铜柱型。

## 五、 接线方式

下图以 RM1702，2 路输入 2 路输出为例。其他 RM17 型号类似。

1. 输入端可以接开关、NPN 输出传感器、PLC 等等，采用共阴极接法。
2. 输出端是继电器。



## 六、 驱动

Windows、Linux、Android、Mac OS 免安装驱动。（Win7 或更低版本需要安装）

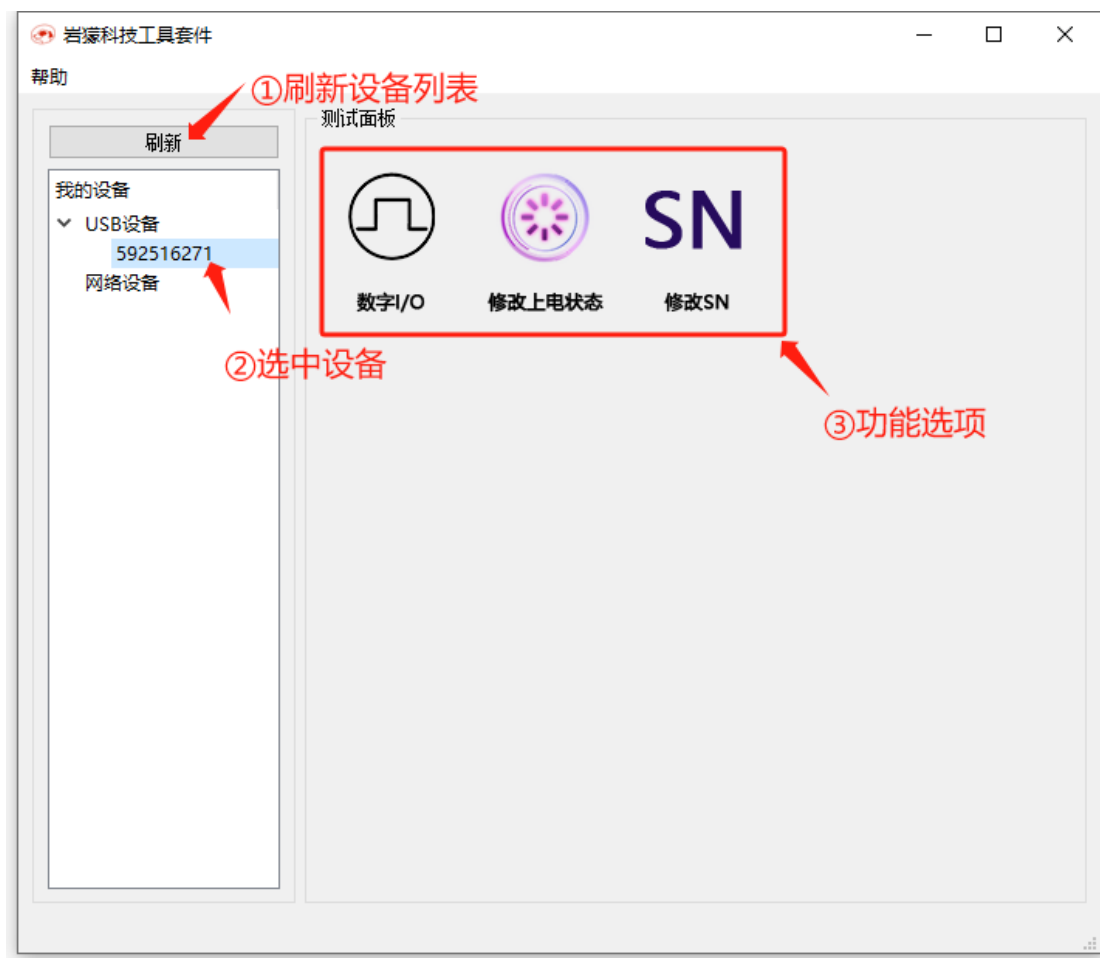
## 七、 调试工具使用介绍

调试工具图标：



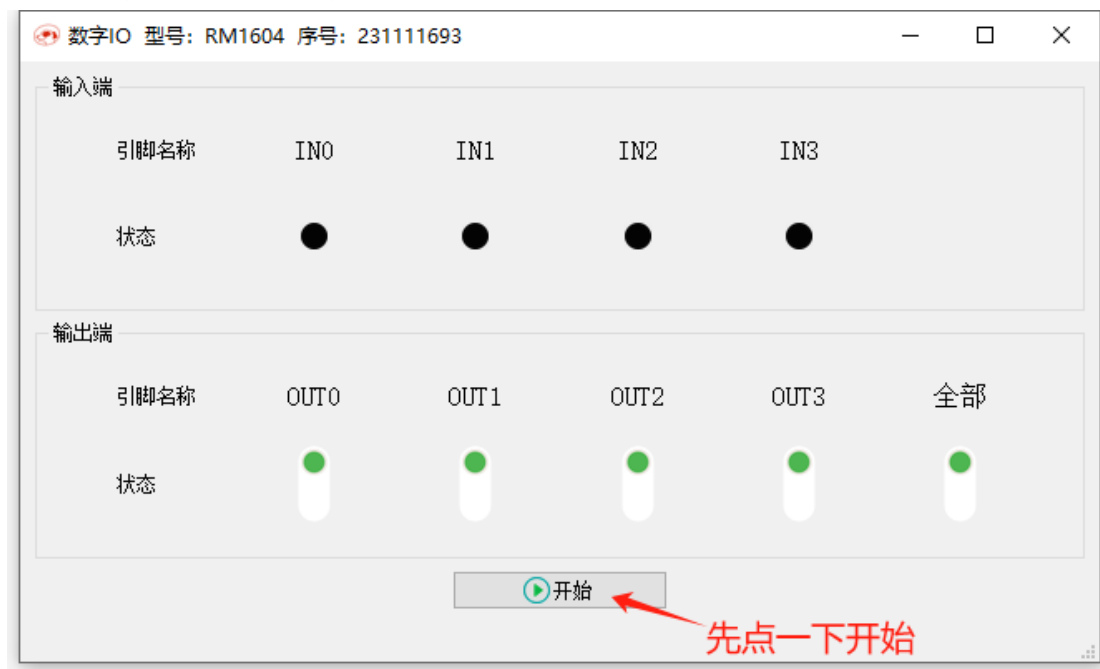
### 1. 工具总览

双击打开后：



## 2. 数字 I/O

用来实时读写 IO。点击“数字 I/O”打开后：



### 2.1 引脚模式介绍

1. 输入模式：用来读取 IO 的电平状态。例如用来识别开关状态、读取传感器输出等等。
2. 输出模式：用来控制输出继电器状态。

### 2.2 状态介绍

1. 输入端：IO 电平状态指示灯。黑色代表低电平，绿色代表高电平。
2. 输出端：继电器控制开关按钮。黑色代表断开，绿色代表吸合。点一下就会翻转状态。

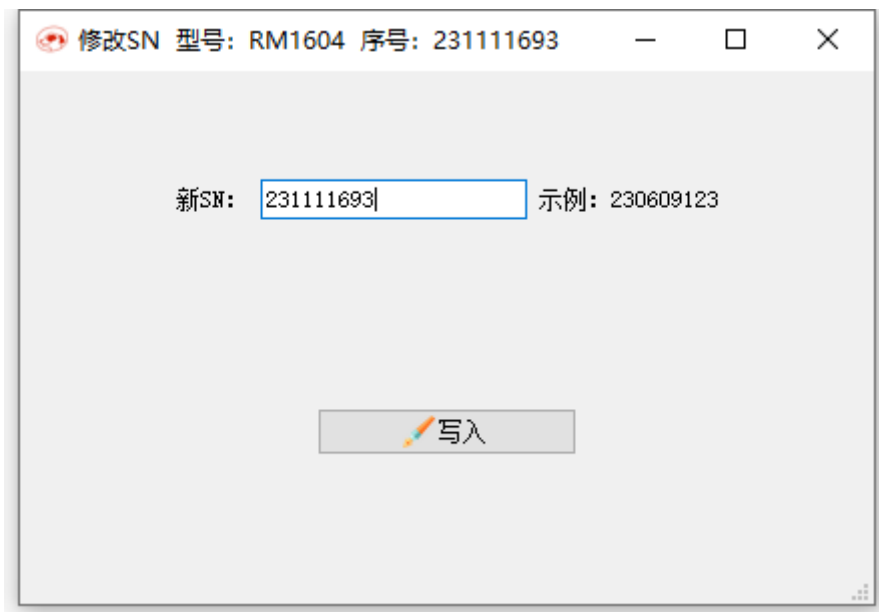
### 3. 修改上电状态

用来修改上电时，继电器的输出状态。状态选项参照“数字 I/O”章节中的描述。修改好后，点击写入按钮。弹出写入成功弹框后，重新插拔 USB 线，即可生效。



### 4. 修改 SN

先填写新的 SN，长度范围 1~9 个数字。写好后点击写入按钮。提示成功后，回到主程序页面，点击刷新按钮刷新设备列表。



## 八、 基础编程一介绍——单个 IO 操作

需要使用到两个库 librockmong 和 libusb-1.0.

这里只介绍 C 语言下的库函数，其他语言类似，具体请参考各语言下的 Demo\_IO 例程。

### 1. 获取设备

1. //扫描 USB 设备，获取设备序列号列表
2. //返回值如果大于 0，代表获取到设备的个数。如果等于 0，代表未插入设备。如果小于 0，代表发生错误
3. `int UsbDevice_Scan(int* SerialNumbers);`

### 2. 读取输入状态

1. //读取引脚状态
2. //SerialNumber: 设备序号
3. //Pin: 引脚编号。0, IN0. 1, IN1...
4. //PinState: 返回引脚状态。0, 低电平。1, 高电平
5. //函数返回: 0, 正常; <0, 异常
6. `int IO_ReadPin(int SerialNumber, int Pin, int *PinState);`

### 3. 控制输出状态

1. //控制引脚输出状态
2. //SerialNumber: 设备序号
3. //Pin: 引脚编号。0, OUT0. 1, OUT1...
4. //PinState: 引脚状态。0, 继电器断开。1, 继电器吸合。
5. //函数返回: 0, 正常; <0, 异常
6. `int IO_WritePin(int SerialNumber, int Pin, int PinState);`
- 7.

### 4. 读取输出状态

1. //读取输出引脚状态
2. //SerialNumber: 设备序号
3. //Pin: 引脚编号。0, OUT0. 1, OUT1...
4. //PinState: 返回引脚状态。0, 继电器断开。1, 继电器吸合。
5. //函数返回: 0, 正常; <0, 异常
6. `int IO_ReadOutputPin(int SerialNumber, int Pin, int *PinState);`

## 九、 基础编程二介绍——多个 IO 同时用 bit 操作

这里只介绍 C 语言下的库函数，其他语言类似，具体请参考各语言下的 Demo\_IO\_Bit 例程。

### 1. 读取输入状态

```
1. //同时读取所有输入引脚状态
2. //SerialNumber: 设备序号
3. //PinState: 返回引脚状态。每一个 bit 代表一个 IO。如 bit0 为 IN0, bit1 为 IN1, 以此类推
4. //      相应 bit 为 0, 低电平。1, 高电平
5. //函数返回: 0, 正常; <0, 异常
6. int IO_ReadPin_Bit(int SerialNumber, int* PinState);
```

### 2. 控制输出状态

```
1. //同时控制所有引脚输出状态
2. //SerialNumber: 设备序号
3. //PinState: 写入引脚状态。每一个 bit 代表一个 IO。如 bit0 为 OUT0, bit1 为 OUT1, 以此类推
4. //      相应 bit 为 0, 继电器断开（晶体管导通）。1, 继电器吸合（晶体管断开）
5. //函数返回: 0, 正常; <0, 异常
6. int IO_WritePin_Bit(int SerialNumber, int PinState);
```

### 3. 读取输出状态

```
1. //同时读取所有输出引脚状态
2. //SerialNumber: 设备序号
3. //PinState: 返回引脚状态。每一个 bit 代表一个 IO。如 bit0 为 OUT0, bit1 为 OUT1, 以此类推
4. //      相应 bit 为 0, 继电器断开（晶体管导通）。1, 继电器吸合（晶体管断开）
5. //函数返回: 0, 正常; <0, 异常
6. int IO_ReadOutputPin_Bit(int SerialNumber, int* PinState);
```

## 十、 其他编程介绍

这里只介绍 C 语言下的库函数，其他语言类似，具体请参考各语言下的 Demo\_IO\_Multi 例程。

### 1. 多个输入口同时读取

```
1. typedef struct
2. {
3.     uint8_t Pin;    //引脚编号
4. }IO_Read_Struct_Tx_t;
5.
6. typedef struct
7. {
8.     uint8_t Ret;    //返回: 0, 正常; <0, 异常
9.     uint8_t PinState; //引脚状态
10. }IO_Read_Struct_Rx_t;
11.
12. //同时读取多个输入口状态
13. //SerialNumber: 设备序号
14. //TxStruct: 发送数据结构体指针
15. //RxStruct: 接收数据结构体指针
16. //Number: 结构体的个数
17. //函数返回: 0, 全部正常; <0, 存在异常
18. int IO_ReadMultiPin(int SerialNumber, IO_Read_Struct_Tx_t* TxStruct, IO_Read_Struct_Rx_t* RxStruct, int Number);
```

### 2. 多个输出口同时写入

```
1. typedef struct
2. {
3.     uint8_t Pin;    //引脚编号
4.     uint8_t PinState; //引脚状态
5. }IO_Write_Struct_Tx_t;
6.
7. typedef struct
8. {
9.     uint8_t Ret;    //返回: 0, 正常; <0, 异常
10. }IO_Write_Struct_Rx_t;
11.
12. //同时写入多个输入口状态
13. //SerialNumber: 设备序号
```

```
14. //TxStruct: 发送数据结构体指针
15. //RxStruct: 接收数据结构体指针
16. //Number: 结构体的个数
17. //函数返回: 0, 全部正常; <0, 存在异常
18. int IO_WriteMultiPin(int SerialNumber, IO_Write_Struct_Tx_t* TxStruct, IO_Write_Struct_Rx_t* RxStruct, int Number);
```

### 3. 多个输出 IO 同时读取状态

```
1. struct IO_ReadOutput_TxStruct
2. {
3.     uint8_t Pin;
4. };
5. typedef struct IO_ReadOutput_TxStruct IO_ReadOutput_TxStruct_t;
6.
7. struct IO_ReadOutput_RxStruct
8. {
9.     uint8_t Ret;
10.    uint8_t PinState;
11. };
12. typedef struct IO_ReadOutput_RxStruct IO_ReadOutput_RxStruct_t;
13.
int IO_ReadMultiPin(int SerialNumber, IO_ReadStruct_Tx_t* TxStruct, IO_ReadStruct_Rx_t*
```